Princess Sumaya University for Technology
Computer Engineering Dept.
22344 – Microprocessors - Spring 2025
Software Project

Customizable Quiz Game with User Authentication and Dynamic Question Loading in 8086 Microprocessor

---

The objective of this project is to develop a Multiple-Choice Question (MCQ) Quiz Game using 8086 assembly language, emphasizing modular programming, user interaction, and memory manipulation. Each student will implement a customizable and interactive quiz system featuring user authentication, shuffled question selection, and a scoring system.

Specifications and Requirements:

1. **User Authentication and with Personalized User List:**

Implement a simple login mechanism, as follows:

- Create a predefined list of 3 username-password pairs. The usernames must be based on your immediate family members' names.
- Prompt the user to input their credentials (username and password). (Hint: use int 21h/0AH).
- Compare with the stored list and grant or deny access accordingly.
- If login fails after 3 attempts, terminate the program.

2. **Dynamic Question Loading & Customization:**
   - You must create 4 different MCQ questions with 4 answer options (A, B, C, D) each.
   - All questions must be based on a very specific topic that you personally enjoy or find interesting.
   - Clearly mention your chosen topic at the beginning of your report.
   - Ensure that all questions and answers are written in your own words and relevant to the selected topic.
   - Store the questions and answers in memory.
   - At run time, each game session should shuffle the questions. (see below hint for shuffling in 8086)
   - The program displays questions one by one.
   - Answers can be input as either uppercase or lowercase (A or a, B or b, etc.).

3. **User Input:**

- Accept character input for answers (A, B, C, or D) in case-insensitive manner.

- For valid inputs: display a message that the answer is correct.
- For invalid inputs: display a message that the answer is incorrect and generate two beep sounds. (Hint: simply use INT 21h /02)

5. Scoring System

- Give 1 point per correct answer.
- No negative marking.
- Display the final score out of 5 at the end of the quiz.

Bonus feature 1: Countdown Timer with Sound

- Implement a timer for each question using loop counting with delays.
- The duration of the timer must be calculated using your university ID number as follows:
- Multiply the last digit of your student ID by 2

  (e.g., ID: 20180245 → last digit 5 → timer = 10 seconds).

- If the last digit is 0, use the second-last digit instead.
- Display the countdown timer on the screen.
- If time expires:

  1. Automatically skip the question.
  2. Assign 0 score for the unanswered question.

Bonus feature 2: Scoreboard (Leaderboard)

- Maintain a scoreboard of top 3 scores (username + score).
- Update the leaderboard after each game.
- Display the top scores at the end of each session.

Technical Requirements

1. **Code Structure**:

- Organize your program code into a main body and distinct procedures (Login handling, Question display, …)
- Each procedure should perform a specific task.

2. **Clock Cycle Calculation**:

- Select one of your major procedures (clearly state which one).
- Calculate the total clock cycles required for that procedure. using the 8086-instruction timing sheet. (Provided on the eLearning).

- Determine the time consumed by the procedure for an 8086 system with frequency of 5 MHz.
- Provide detailed calculations.

Deliverables:

1. **Code Submission**:
    - Upload your code as a .asm file.
    - Name the file using your name and student ID.
2. **Report Submission**:
    - Write a report that includes:
        - A detailed description of your work.
        - The main steps of your program, supported by a flowchart.
        - Screenshots of the program's output at each stage (login, question display, score,..)
        - Discussion of the procedures.
        - Clock cycle & timing calculations.

Grading Rubric:

Project Requirement	Grade
Functional code, meaningful procedure/label names, and well-organized procedures.	10
Correct handling of user authentication	7
Correct storing of question-and-answer pool	5
Correct loading of the shuffled question	5
Correct final score calculation	3
Accurate calculation of procedure clock cycles and timing.	10
Proper report writing.	10

Total Grade: 50

Important Notes:

- Ensure you fully understand your code.
- Failure to answer questions about your project will result in a zero grade. (Don't justify you not knowing by saying ChatGPT helped me !!)
- Inability to explain any implementation details will result in no gain for any grade in the bonus features.
- Any fully or partially identical implementation and report will result in zero for the two students.

Good Luck

Helper code for shuffling list of numbers based on fixed seed. Modify it to suit your task.

```
Items DB 3, 1, 2        ; Items to shuffle (can be replaced with any data type)
Seed DB 1               ; Initial seed value (can be modified to any fixed number)

ShuffleLoop:
   ; Generate random index (0, 1, 2)
   MOV AL, Seed         ; Use the seed as the current value
   MOV BL, 3            ; Modulo 3 to ensure the index is in range 0–2
   ; Scale the seed and avoid overflow
   MOV AH, 0            ; Clear AH to ensure safe division
   MOV CL, 4            ; Let's scale the value by a small constant
   MUL CL               ; AL * 4 = AL (scaled value)
   MOV BL, 3            ; Now divide by 3
   DIV BL               ; AX / 3 (quotient in AH, remainder in AL)
   MOV DL, AL           ; Random index is now in DL (0–2)

   ; Swap Items[DL] with Items[DL+1] or any two different items
   MOV SI, OFFSET Items  ; Address of Items array

   ; Simple swapping logic for index 0, 1, 2
   CMP DL, 0
   JE Swap00
   CMP DL, 1
   JE Swap01
   CMP DL, 2
   JE Swap02

Swap00:  ; Swap Items[0] and Items[1]
   MOV AL, [SI]         ; Load Item[0] into AL
   MOV BL, [SI + 1]     ; Load Item[1] into BL
   MOV [SI], BL         ; Store BL into Item[0]
   MOV [SI + 1], AL     ; Store AL into Item[1]
   JMP DoneSwap

Swap01:  ; Swap Items[1] and Items[2]
   MOV AL, [SI + 1]     ; Load Item[1] into AL
   MOV BL, [SI + 2]     ; Load Item[2] into BL
   MOV [SI + 1], BL     ; Store BL into Item[1]
   MOV [SI + 2], AL     ; Store AL into Item[2]
   JMP DoneSwap

Swap02:  ; Swap Items[0] and Items[2]
   MOV AL, [SI]         ; Load Item[0] into AL
   MOV BL, [SI + 2]     ; Load Item[2] into BL
   MOV [SI], BL         ; Store BL into Item[0]
   MOV [SI + 2], AL     ; Store AL into Item[2]

DoneSwap:    ; Update the seed for next shuffle
   MOV AL, Seed
   MOV BL, 7
   MUL BL               ; AL = AL * 7 (random multiplier)
   ADD AL, 3            ; Add a constant (e.g., 3)
   MOV Seed, AL         ; Update the seed value
```